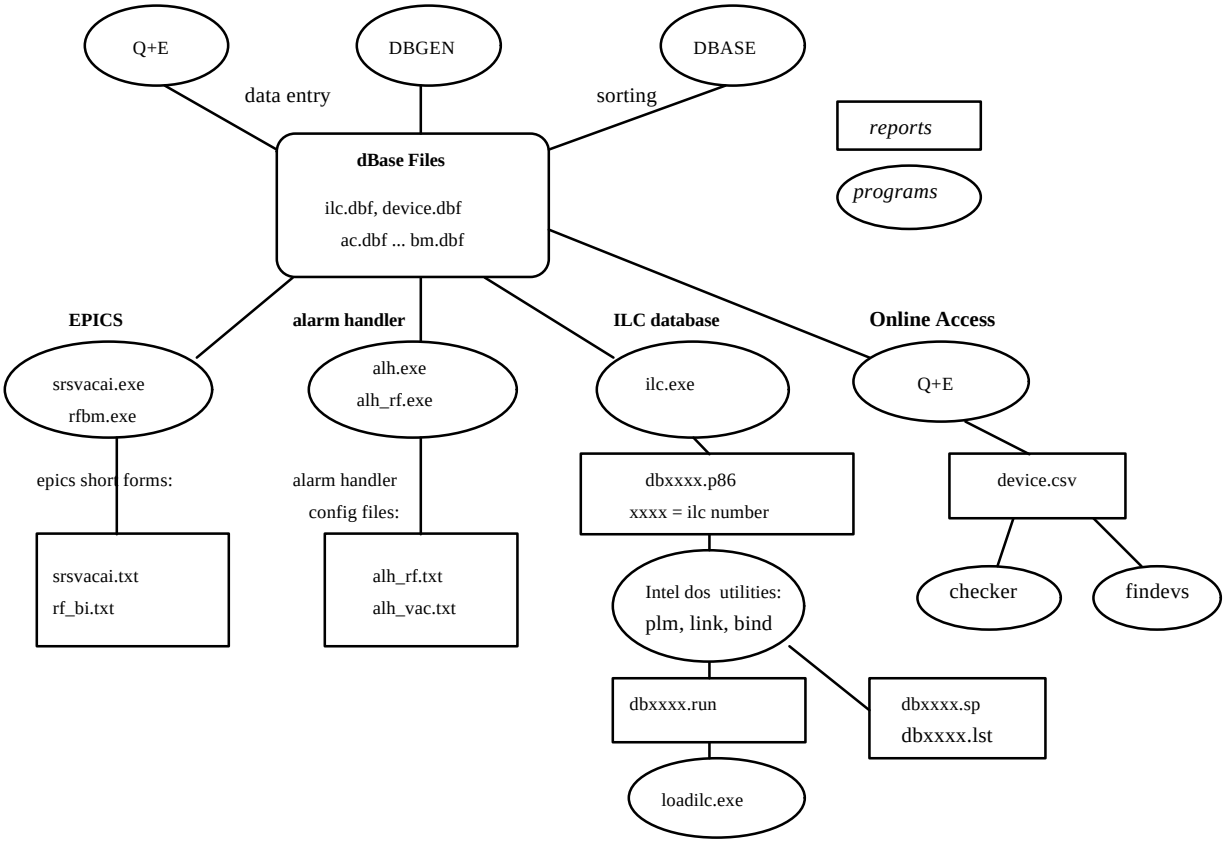
Revision 2 March 15, 1994

ALS Database

Chris Timossi

Database Generation and Reports



This manual was produced using*Doc-To-Help*®, by WexTech Systems, Inc.

WexTech Systems, Inc.
310 Madison Avenue, Suite 905
New York, NY 10017
(212) 949-9595

Contents

Introduction

Purpose

This document will be helpful to two groups: programmers, that need a more complete understanding of the of the ILC database usually because they are programming at the ILC level, and people involved in adding new ILC's or changing existing ILC's to control new hardware. This is not a document to explain how to program an ILC or how to write programs to access the ILC database. The audience for this document is expected to have some familiarity with the general architecture of the control system, i.e. to know what an ILC or CMM is. Also, some of the procedures outlined will assume familiarity with our PC-based network file system.

ALS Database Systems

The term *ALS Database* really refers to two distinct database systems. The first system is called the *CMM/ILC database* (sometimes this system is described as the real-time database). This database describes the structure of data, including data gathered in real-time, in the ILC and CMM memory. This structure is used mainly by programs running in the ILC, CMM, and DMM to control the accelerator instrumentation. The second database system called the *offline database* (sometimes refered to as the DBASE database) is a relational database system, residing on the file server, that contains static data needed to create the CMM/ILC database. This database system is also used to store other engineering parameters that are not used in the ILC database but are related to control of the device; the idea is to keep related control information in one place for the convience of generating reports. These reports range from those that are used to generate the ILC database to those that are used to generate EPICS short forms.

Other References

For information on programming, see the Intelligent Local Controller Manual The ALS Control System Documentation contains is a collection of all the control system documents. The ALS Parameter List contains a useful description of the naming convention at the ALS. On line documentation is available on the file server (including this document) under \docs.

CMM/ILC Database

Introduction

Control of Accelerator Hardware

The CMM/ILC database refers to a collection of data structures in the ILC memory which are used for two distinct purposes. First they contain static data which includes a name that uniquely identifies a database element throughout the control system and also includes any constants or parameters that can be used by a general control program running in the ILC. The idea is to avoid having different code for every ILC. Second, the database structures are used to contain the real-time data gathered by ILC from the accelerator hardware. These structures are located in an 8k block of ILC memory and are duplicated in the CMM memory for each ILC; the CMM contains the complete database for controlling the accelerator. Anything in the accelerator that needs to be monitored or controlled, must go through this database. These data structures are just C or PLM language structures. It is important to realize that the database is arranged in a bottom up fashion: all the data necessary to control and identify a piece of accelerator equipment resides in the ILC database; the CMM database is created from all these individual databases at *boot* time. The ILC can be disconnected from the rest of the system (or the rest of the system can be shut down), but the ILC will continue to control the devices in its database.

Dataflow

Reading data

Data that is to be aquired from the instrumentation, comes into an ILC through several types of channels: ADC, boolean, bit-parallel, serial and GPIB. The ILC software is responsible for depositing this information into a database structure of the appropriate type for any data that needs to be accessible from any other layer of the control system. As the data is being read by the background ILC program from the hardware and being put in the database, copies of these structures are

periodically and asynchronously, sent to the CMM whether or not any data has changed. It is from this complete CMM database that an application that needs it reads the aquired data.

Setting data

Data is sent to the instrumentation from the ILC through the following channels: DAC, boolean, bit-parallel, serial, and GPIB. Any data that must be set from a higher level of the control system, must be eventually be placed in a database structure, of the appropriate type, in the ILC. Typically, data comming from the DMM will be placed in a buffer in the CMM. This data placed in this *send buffer* is periodically sent to the ILC. When the data arrives at the ILC, an interrupt occurs and the handler puts the data in the database structure. Typically, the background (non-interrupt based) program in the ILC, periodically and asynchronously sends the data to the hardware whether or not the data has changed.

Data Structures

Introduction

The database structures are organized on a *channel* basis. The term *channel* refers to a physical data path between the hardware to be controlled and the ILC. For instance, the voltage of a power supply is monitored by an ADC on the ILC, the resulting data is placed in a structure in the memory of an ILC; this path from the hardware to the ILC, via the ADC is refered to as an ADC channel. There are usually three linked database structures for each channel that is being used in the ILC. The DBELEMENT structure is the parent structure and contains the information common to all channels and indices to the two other structures which contain data that is specific to the data or *channel type* (an ADC has a different structure than a DAC for example). The very first entry in the DBELEMENT structure is worth mentioning here: it is an ascii name that is unique to the channel throughout the entire control system. Access to a database channel is based on searches on ascii names contained in a DBELEMENT structure. Additionally, there are channels that are not associated with hardware. One important example of such a pseudo-channelis the device (DV) channel type. Although, in simple cases, controlling accelerator equipment can be done with a few analog and digital channels which are independent of each other, in most cases equipment is controlled by groups of related channels called *devices*. A special data structure called a device channel (DV) is the first structure before a group of such related channels. A special structure that occurs in every ILC is the *last element* structure (LA). Since data is only communicated to or from the ILC by database fields, information about the ILC itself, is put into this special channel type so that it can be monitored by upper layers of the control system.

ILC and CMM Memory Layout

Although the latest ILC has 256k of memory, an ILC database is restricted to an 8k block (0x1000-0x3000). The complete database, in the CMM, maintains an identical block for each ILC in the system. These 8k blocks are divided into 2 areas: the *inactive* area and the *active* area. The inactive area begins at the start of the block and grows (as more devices are added to the ILC) towards upper memory.

The inactive area contains static information such as device names and channel numbers that cannot change dynamically. This area is only copied to its corresponding block in the CMM during a *boot* of the ILC. It can only be changed offline and then *downloaded* to the ILC. At the beginning of the inactive area, is an array of structures of type DBELEMENT; one structure for each *channel* that a particular ILC controls. The DBELEMENT array is followed by an area called the *inactive particular* part. This part consists of static data structures whose type depends on the type of channel. The whole inactive area is only uploaded to the CMM when a boot request is received. The *active* area is also made up of structures whose type depends on the channel. But as the name suggests, all fields in these structures contain dynamic data: any data that needs to be automatically updated in the CMM. This includes data read from the channels controlled by the ILC but also **any** information that needs to be communicated to or from the ILC. As mentioned before, the database structures are linked: DBELEMENT contains two pre-calculated indices, one is an index to a structure, whose type depends on the channel type, in the active area the other is an index to a structure in the active area.

Use of Structures by ILC code

Although one of the main design goals in the development of the ILC software was to make it data-driven, i.e. to have the personality or function of the ILC totally determined by the database that is loaded, in reality there about half a dozen different ILC programs that were written to handle the special needs of the accelerator. ILCSTD is the name of the generic code that handles most needs of the accelerator. IRAMP was written to handle the special requirenments of the booster magnet power supplies that must track the booster bend magnet. IBPM handles the storage ring Beam Position Monitors. Other code is in development to handle undulators and vacuum chamber protection. In most cases all these different programs use that database fields in a consistant way, but there are several potentially confusing issues. Not all programs use every field, in fact, some fields were included to handle the special needs of a particular control program. Some databases contain pseudo-channels. For example, when an ILC controls a GPIB device, analog data is gathered from the device by GPIB commands not by using the ILC's ADC. But an analog database structure is used to store the data collected. Some of the fields in the structure, channel number for instance, are meaningless in this case.

ILC code typically functions in the following fashion: it has a main loop in which it steps through each database element (DBELEMENT). When it encounters an ADC channel type, for example, it reads the analog data from the device and puts it into the active area for that database element. When it encounters an output channel, such as a DAC, it reads from a value from the active area and writes it to the DAC. Both of these steps occur whether or not there is any change in the data. Sometimes, channels cant be processed independently. An example is the handling of the *software seal* When the ILC, in its normal scanning of channels, detects that a boolean monitor of a *ready* channel goes to the not-ready state, it must find the realated control channel to command the device to go off. Simarly, there is code to handle analog control/analog monitor closed loop. The ILC must keep track of realated ADC/DAC pairs to handle this algorithm so special fields in the DAC channel structure were added for this purpose. A more complicated situation occurs when large amounts of data, such as a waveform, is gathered over a fairly low bandwidth channel such as a GPIB interface. In this case, the ILC uses a database field as a kind of data-validsemaphore which is only set when the whole block of

data has been aquired. Programs at higher layers can check this field, by polling, to determine when the data is valid. There are also several examples of ILC code that operates in a 'stateful' manner. The BPM and TIMING ILC code, both poll predefined fields for commands. For example, the BPM code polls a field in the BP structure that is set by a higher level program to tell the ILC to re-arm the FAD.

Header Files

The following is a C language definition of the database structures. These are contained in the header files: **dbpoi.h** and **dbdeftyp.h**. **dbdefine.h** and **hardware.h** contains some manifest constants that are defined for use with the structures in a C program. In addition, the file **dbchan.dec** contains the PLM definition of these same structures. These two structures must match, one should not be changed without changing the other. dbchan.dec also exists on the RMX file system to be used when re-compiling the DMM software. Any change to these structures requires recompiling all ILC code and the DMM code. PC applications are generally uneffected; but if DBELEMENT changes, then the DBACT routine in the linkb library must be compiled. In addition, GETSTRUCT.C should be looked at. Finally, all structures must have lengths divisble by 4 and any word or 4 byte data type (like float) must be on a 4 byte boundry.

Datatypes

The following data types are used in define the database structures and are declared in **ptypes.h**:

DEFINED TYPE	Storage
UBYTE2	2 bytes; unsigned
SBYTE1	1 byte signed; used for chars
UBYTE4	unsigned 4 bytes
REAL4	IEEE single precision (4 bytes)

DBELEMENT Structure

There is one dbelement structure in the inactive part of the database for every channel that is to be controlled by an ILC. This structure contains static data that is the same for any channel regardless of type. In addition, this structure contains an index to an active structure and an index to an inactive structure; these structures contain the type specific information. DBELEMENT contains a name, as it's first entry that is unique throughout the control system. An ILC database contains an array of DBELEMENTS, the*Index* into this array together with the ILC number forms a pair (index-ilc) that uniquely identifies this element (until the ILC database is modified).

SBYTE1 DisplayName[DB_DISPNAMELEN];

The *System Name* followed by a space followed by the *ID name.*

SBYTE1 ChannelTypeName[DB_CHANNAMELEN] ;

A 2 character name, indicating the type of channel followed by two digits indicating the channel number. Examples of the two character channel name are AC for analog control, AM for Analog Monitor and BC for Boolean Control. This part of the name combined with the previous part of the name identifies the channel uniquely throughout the whole database.

SBYTE1 DeviceTypeName[DB_DEVNAMELEN] ;

This part of the name identifies the functional subsystem of the channel. Examples of these subsystems are RF for Radio Frequency, DIAG, for diagnostic and MPS for Magnet Power Supply.

SBYTE1 NetName[DB_NETNAMELEN] ;

Describes the use of the channel.

UBYTE1 ChannelType ;

Provides the same information as the ChannelTypeName, but in numeric form.

UBYTE2 InactiveIndex ;

An offset to the inactive structure for this channel.

UBYTE2 InactiveLength ;

The size of the Inactive structure for this channel.

UBYTE2 ActiveIndex ;

An offset to the active structure for this channel.

UBYTE2 ActiveLength ;

The length of the active structuer.

UBYTE1 FunctionType ;

The original intent of the FunctionType would be that a channel that needs to be related to a follow channel or channels, would be given a non-zero code that would tell the ILC what function it should perform on this group of channels (the following channels would have 0 entries to indicate that they belonged to a previous channel). An expample of such a relation is the 'software seal' algorithm that the ILC uses which causes a boolean control to be reset when the following boolean monitor channel (the *ready* channel) is 0. The role of function type is also, in some cases, played by the Type field in the Device channel structure.

Active Particular Structures

These structures contain data that is specific to a particular *channel type* as entered into the DBELEMENT structure. All active structures begin with the following fields:

Error: when the ILC detects a hardware error such as a value being out of range, it places the code (as defined in errbdec.c) in this field.

PreviousError: when an error is detected by the ilc, any previous value in the Error field is put here; thus, one error back is remembered.

ErrorMask: errors are placed in categories according to severity, this field can be used to mask out errors by category (not implemented).

Permit: occurs only in control active structures. It can be used to disable the control of the channel (partially implemented).

Adjust: is used to make sure the structure is a multiple of 32 and to make sure that 4 byte data types start on 32 bit boundries.

Inactive Particular Structures

All inactive structures begin with these fields:

DevLink: is the number of elements back to the parent Device Channel. The DevLink for a Device Channel is 0, the first channel after it is one, etc. Some devices, such as LTB trim power supplies, are composed of several device channels. The first device channel still has DevLink = 0, but subsequent DevLink values are 1 signaling the ILC code to use the previous channel 's DevLink to find the parent DV.

ChannelNumber: is used to distinguish between one of several channels of the same type. For example, since an ILC has four ADC input channels and four DAC output channels (0-3), the channel number is used to identify one in particular. The ILC also has 24 total single bit (boolean) channels that can be configured, within limits, as either controls or monitors. The channel numbers are always sequential from 0 to 23 whether controls or monitors. There are some structures, such as Device, Mux, and Graphic, that are not associated with real hardware channels. By convention, the ChannelNumber for these structures should still be unique within the ILC database (even though they may not be used).

A description of the rest of the structures follows. The fullname of the structure followed by its 2 character channel name is given. Next, the fields are described. Some descriptions are omitted for those fields that are no longer in use.

Device (DV)

*The **device type** determines the number and type of channels in a device.*

A device channel is a special type of channel that is used to indicate the start of a group of related channels that form a device. The number and types of channels th form a device is determined by the **Type** field. The device channel structure also contains offsets values that can be used to locate certain other commonly used channels in the device.

INACTIVE

UBYTE4 BeamOrder ;

This device's position relative to other devices in the accelerator.

UBYTE4 DefaultDelay ;

Used as a counter for dev. power-up (unused)

UBYTE2 BeamLine ;

Unused

UBYTE2 SubsysMask ;

Unused

UBYTE1 Type ;

Determines channel composition. The ILC uses this field when it has to do something special for a device that may include several channels.

UBYTE1 NormalStatus ;

Nominal operating status is defined as the status of the device during normal operation of the accelerator.

UBYTE1 ACoffset ;

Offsets to dbelement of first AC channel.

UBYTE1 AMoffset ;

Offsets to dbelement of first AMchannel.

UBYTE1 BMoffset ;

Offsets to dbelement of a BMchannel.

UBYTE1 BCoffset ;

Offsets to dbelement of first BCchannel.

UBYTE1 RDYoffset ;

Offset to Ready mon. if present.

UBYTE1 BNFoffset ;

Offset to ON/OFF monitor if present.

UBYTE1 ChainOffset ;

Offset to the first BM in an interlock chain.

UBYTE1 NRchannels ;

Number of channels in this device.

UBYTE1 Status ;

LastElement (LA)

Every ILC has one LastElement channel as the very last channel. The inactive part contains fields that are used in the initialization of the ILC and for consistancy checking. The active area contains fields that are used in link checking and checking the health of the ILC.

INACTIVE

UBYTE2 InactiveLowerLimit ;

UBYTE2 InactiveUpperLimit ;

UBYTE2 ActiveLowerLimit ;

UBYTE2 ActiveUpperLimit ;

The above limits contain the ranges of memory locations that are used for the inactive and active database areas.

SBYTE1 Date[8] ;

Date the ILC was downloaded

SBYTE1 Time[8] ;

Time the ILC was downloaded.

SBYTE1 Name[10] ;

ILC name.

SBYTE1 RunFileName[40] ;

The ILC code file name loaed into the ILC

SBYTE1 VersionNumber[30] ;

The version number of the ILC code.

UBYTE1 OnboardConfig ;

See description under Offline Database.

UBYTE1 ILCType ;

See description under Offline Database.

UBYTE1 SBXConfig ;

See description under Offline Database.

UBYTE1 SBXType ;

The type of SBX module(see description under Offline Database).

ACTIVE LA

UBYTE2 Status ;

The status of the ILC.

UBYTE2 Watchdog ;

A counter incremented by the ILC code once per scan loop.

UBYTE2 LinkTest ;

A counter incremented by the ILC code once per scan loop.

Analog monitors (AM)

This structure is used to store an ADC value and related scaling factors that is read from an on board or SBX mounted ADC. The accuracy of the ADC is given by ADCResolution and is usually 12 or 16 (bits).

INACTIVE AM

REAL4 FSValue ;

Full scale value in engineering units used to scale ADC channel.

REAL4 Offsets ;

Offset in engineering units for devices in which 0 volts does not correspond to zero in engineering units.

REAL4 DefaultUpAlarmLim ;

Alarm Limit in engineering units. Error is set when this limit is exceeded.

REAL4 DefaultLowAlarmLim ;

Alarm Limit in engineering units. Error is set when this limit is exceeded.

REAL4 Stability ;

Indicates the reading-to-reading Stability (% F.S.)

REAL4 Deviation ;

If |AC-AM| > Deviation (%F.S.) then turn off closed loop

UBYTE1 ADCResolution ;

Number of bits: 12 for old ILC, 16 for new ILC

UBYTE1 OutputFormat ;

Two digit hex number: first is field width second is digits after decimal.

UBYTE1 Units ;

Code for units (not used)

SBYTE1 UnitsString[DB_UNITLEN] ;

Default units string

UBYTE1 Displayflag ;

True if primary display channel (unused).

UBYTE1 HardwareType ;

Indicates onboard vs sbx chan type.

REAL4 BIConv[4] ;

field/current conversion for storage ring magnets BIConv[2] is used as the stepsize by ILCMAG program.

ACTIVE AM

REAL4 Value ;

ADC reading in floating point engineering units

REAL4 AMReferenceValue ;

a previous reading; saved for reference.

REAL4 UpAlarmLim ;

The current alarm limit. alarm if Value > UpAlarmLim

REAL4 LowAlarmLim ;

The current low alarm limit. Alarm if Value < LOALARMLIM

REAL4 HighWaterMark ;

Save highest reading

REAL4 LowWaterMark ;

Save lowest reading

UBYTE2 TimeStamp ;

Unused.

SBYTE1 AsciiValue[DB_ASCLEN];

ADC Reading in engineering units converted to ascii (using FORMAT) */

SBYTE1 UnitsString[DB_UNITLEN] ;

Current string being used for units.

Analog controls (AC)

This structure is used to hold values to be set to the ILCs 16 bit DAC. The value is stored in floating point and is converted by the ILC before being sent to the DAC.

INACTIVE AC

REAL4 FSValue ;

Full scale value in engineering units that corresponds to the full 10 Volt output of the DAC.

REAL4 Offsets ;

Offset in engineering units that corresponds to 0v output on the DAC.

REAL4 PowerUpValue ;

Value set on ILC power on or reset of the ILC.

REAL4 UpperLimit ;

Largest value that can be set

REAL4 LowerLimit ;

Smallest value that can be set

REAL4 ClosedLoopTol ;

If |AC-AM| < ClosedLoopTolERANCE (%F.S.) closed loop off

UBYTE1 Units ;

/* Device Units: volts; amps; etc. */

SBYTE1 UnitsString[DB_UNITLEN] ;

/* Default Units */

UBYTE1 DACResolution ;

/* Number of bits 12; 16; etc. */

UBYTE1 OutputFormat ;

/* Two digit hex: first field width, second dec. places */

UBYTE1 DefaultKnobScale ;

/* Default knob sensitivity 1-15 */

UBYTE1 HardwareType ;

/* Indicates onboard vs sbx chan type */

UBYTE1 Displayflag ;

/* True if primary display chan */

REAL4 BIConv[4] ;

Field/current conversion. BIConv[2] contains step size. 0 means dont ramp.

The following fields are used and set by the closed loop algorithm

REAL4 PrevValue ;

Value from last DAC reading; used by closed loop algorithm.

REAL4 PrevSPValue ;

REAL4 PrevOutValue ;

REAL4 PrevAMValue ;

void far *AMActPtr ;

void far *AMInActPtr ;

REAL4 AMAverage ;

REAL4 Scale ;

REAL4 DeltaAC ;

UBYTE1 AMAverageCount ;

UBYTE4 Timer ;

UBYTE1 SlewTime ;

Time (in pulses) closed loop is turned off after SP change

ACTIVE AC

REAL4 Value ;

Analog Control setting in engineering units.

REAL4 SPReferenceValue;

The previous control setting.

UBYTE1 ClosedLoop ;

Closed loop on/off control: 255 turns closed loop algorithm on.

UBYTE1 KnobScale ;

The knob sensitivity.

SBYTE1 AsciiValue[DB_ASCLEN];

Fixed point ascii string of Value.

SBYTE1 UnitsString[DB_UNITLEN] ;

Current units.

Byte wide monitor (DI)

This structure is used to input either 8 or 16 bit parallel data.

INACTIVE DI

UBYTE2 NominalState ;

The normal operating value of DMValue.

UBYTE1 ChannelSize ;

Width of the DI input 8 bits or 16 bits

UBYTE1 HardwareType ;

Indicates onboard vs sbx chan type

ACTIVE DI

UBYTE2 PreviousValue ;

The previous reading.

UBYTE2 DMValue ;

The latest reading.

Byte wide control (DO)

This structure is used for 8 or 16 bit output.

INACTIVE DO

UBYTE2 DefaultVal ;

The startup value to output.

UBYTE1 StrobeEnable ;

Use strobed output if true.

UBYTE1 ChannelSize ;

Use 8 or 16 bit input.

UBYTE1 DatumSize ;

Use 8 or 16 bit input.

UBYTE1 HardwareType ;

Indicates onboard vs sbx chan type

ACTIVE DO

UBYTE2 PreviousValue ;

UBYTE2 DCValue ;

Boolean monitor (BM)

This structure is used for single bit monitoring. The ILC uses the values 0 and 255 for the two states. The 255 value means the ILC is reading a high from an opto-isolated input. The channel also has an Operating Statewhich is the state the monitor should have when the accelerator is operating normally.

INACTIVE BM

UBYTE1 OperatingState ;

The value of the BMValue under normal operating conditions.

SBYTE1 LowText[5] ;

Text describing 0 state. (opto low).

SBYTE1 HighText[5] ;

Text describing state with value 255.

UBYTE1 ChainNumber ;

0 if not part of a chain (not used).

UBYTE1 HardwareType ;

Indicates onboard vs sbx chan type.

UBYTE1 DisplayFlag ;

True if this boolean is used as the primary display channel. In a device with many booleans, this one is used to represent the primary state of the device (not used).

ACTIVE BM

SBYTE1 BMText[5] ;

A string that is used to show the current state of the channel.

UBYTE1 BMValue ;

The current state of the channel: 0 or 255. With devices connected to opto-isolators, 255 means the opto is turned on.

UBYTE1 BMReferenceValue;

The previous reading of BMValue.

UBYTE1 ChainNumber ;

Not used.

UBYTE1 Function ;

A code to designate the function of the boolean (e.g. ready, on/off monitor)- not used.

Boolean control (BC)

This structure is used for single bit control. The ILC uses the values 0 and 255 for the two states. The 255 value means the ILC is outputing 0 volts to it's open collector driver which turns the opto-isolator ON. The channel also has an Operating State which is the state the monitor should have when the accelerator is operating normally.

INACTIVE BC

UBYTE1 PowerOnState ;

BC channel goes to 0 or 255 when ILC power is turned on.

UBYTE1 OperatingState ;

State during normal operations.

SBYTE1 LowText[5] ;

Text describing 0 state.

SBYTE1 HighText[5] ;

Text describing 255 state.

UBYTE1 ChainNumber ;

0 if not part of a chain (not used)

UBYTE1 HardwareType ;

Indicates onboard vs sbx chan type

UBYTE1 DisplayFlag ;

True if primary display channel

UBYTE1 PrevValue ;

Previous BCValue reading */

ACTIVE BC

UBYTE1 BCValue ;

The output value (0 or 255)

SBYTE1 BCText[5] ;

Text describing the output value.

UBYTE1 BCReferenceValue;

A value to be used as reference.

Beam Position Monitor (BP)

This structure is for used to contain BPM specific info. These structures are used by the ILCBPM (ILCBPMN) ILC programs for communicating with the BPM electronics. Although data fields are here to customize offsets for individual BPMs, the BPM software does not use them; it currently uses default values in the code.

INACTIVE BP

REAL4 DACFSValue ;

Full scale value in engineering units.

REAL4 PowerUpValue ;

Value set to the DAC on ILC power on

REAL4 UpperLimit ;

Largest value that can be set

REAL4 LowerLimit ;

Smallest value that can be set

REAL4 AMFSValue ;

Full scale value in engineering units.

REAL4 DefaultUpAlarmLim ;

Error field is set when reading is larger than this

REAL4 DefaultLowAlarmLim ;

REAL4 RF1 ;

REAL4 RF2 ;

REAL4 RF3 ;

REAL4 RF4 ;

REAL4 Cal1 ;

REAL4 Cal2 ;

REAL4 Cal3 ;

REAL4 Cal4 ;

REAL4 Xsense ;

REAL4 Ysense ;

UBYTE1 ADCResolution ;

Number of bits 12, 16.

UBYTE1 BPMType ;

The type of BPM: Linac, booster or storage ring. Storage ring bpm uses FADs.

ACTIVE BP

UBYTE1 ActiveFlag ;

BPM active.

UBYTE1 NAvg ;

UBYTE2 TimeStampIn ;

REAL4 AGCValue ;

The range of the BPM.

REAL4 Chan1 ;

Raw signal value from button 1.

REAL4 Chan2 ;

Raw signal value from button 2.

REAL4 Chan3 ;

Raw signal value from button 4.

REAL4 Chan4 ;

Raw signal value from button 4.

REAL4 Average ;

The average reading of the 4 buttons.

REAL4 XPosition ;

The x position of the beam in millimeters.

REAL4 YPosition ;

The y position of the beam in millimeters.

UBYTE1 Status ;

UBYTE2 TimeStampOut ;

SBX GPIB & Serial access. (SC)

This channel type is used for control of GPIB devices and for the new ILC's serial interface.

INACTIVE

UBYTE4 Timeout ;

UBYTE2 EOS ;

UBYTE1 GPIBAddr ;

GPIB or Daisy Chain Address

UBYTE1 OldMessFlag ;

UBYTE1 ScopeDelay ;

ACTIVE

UBYTE1 MessFlag ;

UBYTE1 NumMessBytes ;

UBYTE1 MessageIn[DB_MESSLEN] ;

UBYTE2 CommandCode ;

Analog Test Signal (AT)

These channels are used to name sources of real time signals for scope input.

REAL4 VertScale ;

Vertical label scaling

SBYTE1 ConnectSys[DB_SYSNAMELEN+1] ;

The database name of the MI channel or other measurement device channel to which this AT connects.

SBYTE1 ConnectID[DB_IDNAMELEN] ;

SBYTE1 ConnectChan[DB_CHANNAMELEN] ;

SBYTE1 VertSetUp[DB_VERTCOULEN] ;

Initial vertical settings.

SBYTE1 HorzSetUp[DB_HORZSWELEN];

Initial horz settings.

SBYTE1 EngUnits[DB_UNITLEN] ;

Engineering units label

Mux Input control (MI)

This structure holds Multiplexor Input information

UBYTE1 MuxType ;

Only HP3488 mux so far.

UBYTE1 MuxNumber ;

Which mux mainframe.

UBYTE1 NumberOfOutputs ;

How many outputs it can connect to.

UBYTE1 OldMessFlag ;

Flag holder for communications.

SBYTE1 SaveSet[DB_MUXSAVELEN] ;

String to save setting is mux mainframe.

SBYTE1 MuxSetUp[DB_MUXSETLEN] ;

String setting to close input.

SBYTE1 ConnectSys[DB_SYSNAMELEN+1] ;

The database name of the connect to first MO output, usually the last.

SBYTE1 ConnectID[DB_IDNAMELEN] ;

SBYTE1 ConnectChan[DB_CHANNAMELEN] ;

UBYTE2 CommandCode ;

/* what to do, generally close switch */

UBYTE1 MessFlag ;

/* message handshake */

UBYTE1 InputChannelNum ;

/* corresponding mux input# , filled only on crosspoints*/

UBYTE1 OutputChannelNum ;

/* ditto as input, usually notused*/

Mux output (MO)

Just a way of keeping a linked list of crosspoints.

SBYTE1 ConnectSys[DB_SYSNAMELEN+1] ;

/* either MI, or measurement device */

SBYTE1 ConnectID[DB_IDNAMELEN] ;

SBYTE1 ConnectChan[DB_CHANNAMELEN] ;

SBYTE1 NextSys[DB_SYSNAMELEN+1];

/* Next MO, if any */

SBYTE1 NextID[DB_IDNAMELEN] ;

SBYTE1 NextChan[DB_CHANNAMELEN] ;

SBYTE1 MuxSetUp[DB_MUXSETLEN] ;

/* string setting to close output */

Graphic (GR)

This structure is used to hold arrays of values for various devices. It was orginally designed to handle scope traces, so many fields relate to that particular use. The amount of space reserved in the active database for the array *WaveArray* is determined by the value in *ArraySize*.

UBYTE1 DefaultName[16] ;

UBYTE1 InstrChannel ;

For multichannel instruments.

UBYTE1 OldMessFlag ;

Flag holder for communications.

UBYTE2 ArraySize;

The size, in bytes, reserved for WaveArray.

UBYTE1 OKtoRead ;

Handshake (not used).

UBYTE1 OKtoWrite ;

Handshake (not used).

UBYTE2 UpdateFrequency ;

How often server will inc the UpdateCount. (not used)

UBYTE4 UpdateCount ;

Server inc's this when all data written.

SBYTE1 RWaccess ;

Test&set var to synch readers and writer - not used.

SBYTE1 GRaccess ;

Test&set var to synch GR struct allocation - not used.

SBYTE1 DdeName[DB_DDENAMELEN] ;

eg \\light9\ctlplay|system!GTL BC1AM00 - not used

SBYTE1 HorzLabel[DB_LABELLEN] ;

Sweep speed etc.

SBYTE2 VertPos ;

Vertical trace position.

REAL4 VertScale ;

SBYTE1 VertLabel[DB_LABELLEN] ;

SBYTE1 Title[DB_FULLNAMELEN] ;

UBYTE2 CommandCode ;

What to do, generally gain or sweep speed.

UBYTE1 MessFlag ;

Message handshake.

SBYTE1 MessageIn[DB_VERTCOULEN+DB_HORZSWELEN]

Initiallization string, usually from MI channel

SBYTE1 EngUnits[DB_UNITLEN] ;

Physical unit label

UBYTE2 NumberOfSamples ;

Per graph

REAL4 MinX ;

Scale info.

REAL4 MinY ;

Scale info

REAL4 MaxX ;

Scale info.

REAL4 MaxY ;

Scale info.

UBYTE2 DataType ;

in WaveArray

SBYTE1 WaveArray[1] ;

Continuous data, fake size of 1, usually bigger.

Offline Database

Introduction

The Offline Database is used to contain any data about devices or channels that is useful for control purposes. The idea is to have one central place for such data. The structure of the database can be changed to accomodate new types of information, new data can be entered, interactive queries can be made, and reports can be generated using a commercial database package (see title page diagram). A goal of the design was to allow the Engineering staff to directly enter the data needed by the ILC to control the device that they had designed. DBGEN, a data entry program written in the xbase language and compiled using the Clipper dbase compiler, can be used for this purpose. Once data is entered into the database, interactive queries can be made using the Q&EDatabase Editor. Reports are generated from the database for a variety of uses: to generate the ILC inactive database for downloading into the ILC, to produce EPICS short forms for consumption by the EPICS database entry tool DCTand to produce Alarm Handlerconfiguration files. Also, Q&E is used to produce the file DEVICE.CSV which contains a list of all the devices in the accelerator. This list is used by programs running on the consoles that need to know all the devices that should be active (as opposed to the devices that are currently in the CMM database). Since these reports are run manually, there is no mechanism to insure that all the reports ar e generated whenever a database change is made.

Organization

The Offline database is composed of a set of database files in the dbase file format.Generally, there are two files for each inactive database structure: one contains the inactive data and related information, another contains the default data to be used for each field. For example, the file AC.DBF contains a record for every inactive AC channel in the database. DEFAC.DBF contains one record with default values for an AC record. In addition to the channel specific databases are and ilc dabase (ILC.DBF) that has a record for every ILC and a device database (DEVICE.DBF) that has a record for every device in the database. The device and channel specific databases are related by the fields specifing the *ilc number* and the

beam order. The ilc number is just a number from 1-999 used to address the ilc, every database record in each of the above databases has a field containing this number. The beam order is a number that identifies the physical position of a device based on its distance along the vacuum chamber from the electron gun. There is a beam_order field in the device and channel databases; records with the same beam_order entry are in the same device. Also key to the definition of a device, is the *device type*. The device type tells the number and type of channels that make up the device; this information is stored the DEVTYPE.DBF. The device type number is related to a PLM constant (defined in the dbchan.dec file) in the VALID.DBF database. VALID contains 2 distinct types of data: it contains data that is entered into the device inactive structure, such as device type, and it contains the set of valid entries for various other database fields. These valid entries are used by the database generation code DBGEN to validate user entries.

ilc.dbf

This database contains a record for every ILC in the accelerator. These records contain fields that are pertinent to the ILC itself such as the ILC number and the name of the code to be downloaded. There are two types of ILCs, old and new. The new ILCs are identified by a 'II' written on the handle. A new ILC has and entry in the user_code field that ends in 'n' (ilcstdn.run for example) and has an entry for the SN (Serial Number) field which a number also marked on the handle of the ILC. Old code loaded into a new ILC or new code loaded in an old ILC will cause the ILC not to work.This datbase should periodically be sorted on ILC number.

number

This is the ILC number: 1-999. The numbers 1-63 are reserved for control system work. Also, the numbers are grouped by subsystem:

Subsystem	First ILC	Last ILC
Injection	64	249
Booster	250	344
BTS	350	424
SR	425	799
BL	800	999

name

The name of the ILC is derived from the first device controlled by the ILC.

user_code

The name of the user code to be downloaded into the ILC (see caution above in Overview). Typical names are ILCBPM.RUN (ILCBPMN.RUN) and ILCSTD.RUN (ILCSTDN.RUN). The name must start with an *I* and have only seven characters before the decimal (see loadilc.exe).

IOCONFIG

This entry tells how the parallel ports on the ILC should be setup.

IOCONFIG / IO:	0-7	8-15	16-19	20-23
0	in	in	out	out
1	bpm			
2	in	out	out	in
3	in	out	in	out
4	in	in	in	out
5	in	in	out	in
6	in	in	in	in
7	in	out	out	out

ILC_TYPE

Not used currently. It is meant to be used to distinguish between new and old ilcs.

SBX_TYPE

This field indicates the type of SBX module on the ILC:

SBX_TYPE		Board Type
0		No SBX
1		Robotrol DAC/ADC
2		GPIB
3		Parallel I/O
4		Serial interface

SBX_CONFIG

Currently, this is only used by Linac BPM ILC code to configure the SBX Parallel I/O board. For this configuration, the entry is 7 otherwise it's 0.

SN

Serial number. For new ILC's, indicated by *II-serial number*, the serial number is recorded here.

Device.dbf

Device.dbf contains one record for every *device* in the database. This database should be periodically sorted on beam_order.

ILC

The ilc number: 1-999. If this number is 0, it indicates that it is a known device but that no ILC has been configured to control the device.

SYS_NAME

This string identifies the accelerator system (EG, GTL, LTB, BR, BTS, SR, BL) to which the device belongs. The list of valid entries for this name should be maintained in the VALID.DBF database.

ID_NAME

This string identifies the kind of device being controlled. The kinds of devices are predefined in a combination of places: in the ALS Parameter List and in a series of Memos on the Storage Ring Naming Conventions by Tom Henderson. A typical example of a name is QD1.1 indicating a quad defocusing power supply. The name entered in this field must conform to the standards set in the above sources.

SUBSYS

SUBSYS is the subsystem name. These are predefined (in VALID.DBF) names that indicate the type of system that the device belongs to. Examples entries are MPS for Magnet Power Supply, DIAG for beam Diagnostics, and VAC for VACuum devices.

DEV_TYPE

The DEV_TYPE field is the device type of the device. This number is taken from the DEV_TYPE field of DEVTYPE.DBF and indicates the number and types of channels in the device.

CHAN_NUMB

The CHAN_NUMB field gives the channel number for the device channel type. This number is arbitrary since a device channel doesn't connect to hardware. It is usually defaulted to 0, though some ILC programs use it for their own purposes.

DEV_LINK

This field is not used. The DEV_LINK field in the inactive particular structures is generated by the database generation routine, it is not taken from this database.

DEV_DESCR

This is a string that explains the function of the device. Usually the engineer of the device specifies this description.

BEAM_ORDER

The beam_order specifies the relative position, in beam line order starting at the electron gun, of this device. Part of adding a new device to this database is determining this number based on the physical position of the devices on either side of this device. The range of the beam_order number is determined by the subsystem: EG, LN, LTB, etc. The ranges of beam_order are given in BLSECT.H.

DEVTYPE.DBF

This database contains records for every device type. For example, for a device type that uses 3 channels, there are 3 records, one for each channel in the device. For each device type, there is also a record in the VALID.DBF database that contains the information that has to be loaded in the inactive structure of the device channel.

DEV_TYPE

DEV_TYPE is the device type. There are as many records with the same DEV_TYPE entry as there are the number of channels in the device.

CHAN

The 2 character channel name describing the type of channel.

COMMENT

A string describing the function of the channel in the device.

ORDER

For boolean channels, the order in which the channels appear in the ILC database is crucial. For example, for the software seal, the ready monitor must always come before the on/off monitor. The order entry determines the order in which the boolean monitors will appear in the ILC dataase.

FUNC_TYPE

See the explanation under the description of the DBELEMENT structure.

DUPLICATE

Sometimes a device contains a variable number of channels of a given channel type. The number of channels will be determined by the data enry routine if this field is set to *T*.

HARDTYPE

Since ILCs are configurable with SBX modules, this field tells, for a given channel, where the I/O comes from. For an entry of ONBOARD, the ILC uses its local I/O channels (4 DACS, 4ADCS, 24 BC/BM). The valid entries are enumerated in HARDWARE.H.

DEV_ORDER

A few devices are made up of multiple device channels, in those cases, this field is used to identify the device channel number.

UNIT_T

For boolean channels, this text determines what will be displayed for an active ILC database value of 255.

UNIT_F

For boolean channels, this text determines what will be displayed for an active ILC database value of 0.

OPERATE

When the accelerator is operating normally, the boolean channel should have this value.

Valid.dbf

This database contains data for two different purposes, and should logically be thought of as two different databases. First it contains fields that are used for validation of data entry through the DBGEN database entry program. Next it contains fields that are needed for the device channel's inactive database. For data validation, the columns CSYS_NAMES through CDEV_COM are used. The fields CDEV_TYPE through CHAIN_OFFS are used to generate device channel. See the description of the device database structure for more information about the fields. A description of the confusing fields is given here

CSYS_NAMES - CDEV_COM

These just contain valid entries for SYS_NAME etc. as previously described.

CDEV_TYPE

This number corresponds to the DEV_TYPE field found in the DEVTYP.DBF database. The following columns all relate to this device type. The ILC_CODE entry is never used. The code that is loaded into the ILC is determined in ILC.DBF.

SDEV_TYPE

The PLM literal, defined in DBCHAN.DEC, is looked up based on the entry in the previous column, CDEV_TYPE, when the ILC database is being generated.

SUB_SYS

Not used. The subsystem is determined by the DEVICE.DBF database.

NO_CHANS - CHAIN_OFFS

These fields are entered into the device channel structure. See the description for the device channel inactive database.

AM.DBF

The Analog Monitor database contains a record for every analog monitor channel in the control system. The fields are described under the section on the AM structure. This database also contains fields corresponding to the EPICS Analog Input database fields so that this database can be used to generate an EPICS database.

AC.DBF

The Analog Control database contains a record for every analog control channel in the control system. The fields are described under the section on the AC structure.

BM.DBF

The Boolean Monitor database contains a record for every boolean monitor channel in the control system. The fields are described under the section on the BM structure.

BC.DBF

The Boolean Control database contains a record for every boolean control channel in the control system. The fields are described under the section on the BC structure.

DO.DBF

The Digital Output database contains a record for every digital output channel in the control system. The fields are described under the section on the DO structure.

DI.DBF

The Digital Input database contains a record for every digital input channel in the control system. The fields are described under the section on the DI structure.

BP.DBF

The Beam Position monitor database contains a record for every beam position monitor channel in the control system. The fields are described under the section on the BP structure.

GR.DBF

The GRaphics database contains a record for every graphics channel in the control system. The fields are described under the section on the GR structure.

SC.DBF

The SCope database contains a record for every GPIB or serial channel in the control system. The fields are described under the section on the SC structure.

AT.DBF

The Analog Trace database contains a record for every analog trace channel in the control system. The fields are described under the section on the AT structure.

MI.DBF

The Multiplexer Input database contains a record for every multiplexer input channel in the control system. The fields are described under the section on the MI structure.

MO.DBF

The Multiplexor Output database contains a record for every Multiplexer Output channel in the control system. The fields are described under the section on the MO structure.

Generating an ILC Database

Overview

Generating an ILC database refers to the process of producing a binary file suitable for downloading into an ILC. Database generation is done either because a new ILC database must be created (a new ILC is added to the system) or an existing ILC database must be modified, for example an upper limit field in the ILC needs to be changed. As shown in the diagram on the title page of this document, generating an ILC datatbase file involves several steps. First the information is entered into the Offline database. Next, the ILC.EXEprogram is run specifing the number of the ILC whose database is to be generated. The result is a report in the form of a PLM source code file. This file is compiled, linked and bound into another program, DBMAIN, which is then run to produce the file DBXXXX.RUN (XXXX is the ILC number with leading zeros). This*run* file is then down-loaded to the ILC using the LOADILC.EXE utility.*Updating* an ILC refers to the above steps of running ILC.EXE, running DBMAIN, and downloading the resulting file. Depending on the extent of the database change, operations, engineering or control system staff will be involved in changing and regenerating an ILC database. For simple changes, a data entry program, DBGEN, was written in the dBASE language to allow adding devices or changing fields and to provide data validation. DBGEN is typically used by engineering staff when a new device is being controlled by an ILC. For more extensive changes: modifing the database structure, deleting or moving devices in ILC's, lower level commercial database packages such as dBASE or, more commonly, the Q&E Database Editor, are used by the Controls Group.

Changing the Offline Database

The Database Creation and Modification Cycle

The process of creating and modifying an ILC database is summarized below. To perform these steps requires some experience with DOS and Windows and a 'logon account'. To perform any of these steps requires that user be logged on.

- DBGEN or Q&E is run to make the changes.
- If the ILC is already in the system, UPDATE ILC is done from a control room console and the ILC is re-booted from CTLPLAY; this step completes the changes.
- If this is a brand new ILC, the ILC is placed in a chassis with a special ILC test box connected. The test box allows the I/O channels to be changed and monitored to verify that the ILC is behaving as expected.
- A PC is connected to the ILC and UPDATE ILC is executed. The ILC now contains the new database. After testing, the ILC is ready to be put into the machine and run LOCALLY.
- Changes to the DMM are made to allow the ILC to be run REMOTELY. At this point, further changes to the ILC database may be done from the control room or the ILC may be removed and put into the test setup again.

Using DBGEN

As mentioned above, simple changes to the database can be made using an interactive program called DBGEN. These changes include, changing analog limits, adding a device to an existing ILC, and creating a new ILC database. Using DBGEN requires some preparation be done by the controls group for a new DBGEN user: a sub directory with the user's name must be created under the \ilc\dbase\progs directory and the user must be in the OPDEVor CONTROLS*group* to have permission to write to the relevant directories. Then DBGEN is invoked from the \ilc\dbase\progs directory by: DBGEN <user name>. The user is given a choice of which section of the accelerator to edit. After choosing the section, a list of ILCs is shown.. An existing ILC is chosen from the list when a device has to be added or changed. If a brand new ILC is to be added,*new ILC* is chosen (this choice is at the end of the list) and DBGEN will assign an ILC number and display a list of options for the configuration of the code, parallel I/O and SBX module if present. For either a new or existing ILC, a choice of editing an existing device or adding a new one is presented. Adding a new device brings up a list of devices not yet assigned to an ILC. If the desired device is not in this list of unassigned devices, the new device must first be added to the DEVICE.DBF by the control staff using an ILC number of 0 and the correct beam_order number. Once a device is selected, a list of channels in the device is displayed, selecting one of the channels brings up the appropriate edit screen. After all changes have been made, the database can be saved. No changes are made to the databases until the program is instructed to save them.

Adding Control for a New Device

The process of adding control of a new ILC Database is usually initiated by the engineering staff. Typically a print is available that shows how the device is wired to an ILC. To create a database the following questions must be answered.

What is the device type (what channels does it use) ?

This is usually the hardest question to answer and requires the most experience by the control system person. This question can easily be answered if it is the same type of device as an existing one (just copy the device type). Otherwise, it is up to the control system person to group the channels into existing device types or even create a new device type. Grouping channels into devices requires a few rules to be followed. A device can not have more than one of the following channels: AC,AM, and BC. A device must fit in a single ILC. A device should have a status boolean monitor (ready); it is very convenient if this ready line be the summary state of the device. Obviously, the channels should be logically related.

Where is the device located (what is its beam_order) ?

The beam order must be determined by finding out which device is immediately before the new device and which device is immediately after it.

What kind of device is it (what is its ID_NAME) ?

It is very important to use a consistent naming scheme. Names for a wide variety of devices have been predefined (all defocusing quads always start with QD for instance). Mechanical Engineering should first be consulted if a new name ID_NAME is needed, then it should be added to the ALS parameter list.

Editing the Database Using Q&E

Once the above questions have been answered, the new devices are added to the DEVICE.DBF database. An ILC number of 0 is added if dbgen is to be used. If Q&E is to be used (the database is being created by hand), a real ILC number is picked based on the criteria under the description of ILC.DBF. Now each channel database must be edited to add the SYS_NAME, ID_NAME, ILC, and BEAM_ORDER; these must have the same values as the fields in DEVICE.DBF. The channel databases to change are given in DEVTYPE.DBF for a given device type. A record in ILC.DBF must be added if a new ILC database is being created for this device. The name field should be like the name of the first device in the ILC database; the other fields must have values using the information given under the description of ILC.DBF. If a new device type is added, then it must be added to DEVTYPE.DBF, VALID.DBF and the header files dbchan.dec and dbdefines.h. Typically, adding a device type means changing ILC code to recognize the new type.

Database Utilities

ILC.EXE

The ILC.EXE takes the ilc number (xxxx) as a command line argument and creates PLM source code (dbxxxx.p86) from the Offline Database. This utility is run after changes are made to the database. The PLM source code is copied to the \ilc\dbase\src directory. This utility is usually run as part of the UPDATE batch file. This program was created with the Clipper dBASE compiler. The current version is on \ilc\dbase.

LOADILC.EXE (LOADILC.PIF)

This utility is invoked with an ilc number as an argument optionally followed by a database name (dbxxxx.run), ilc user code file name (ixxxxxx.run), and pc user code (pcuser.run). It loads the files into the ilc. If only the ilc number is given, LOADILC looks in the sub directories db_user, ilc_user, and pc_user for the files to load. It expects ilc code file name to start with 'i' and to be 7 or less characters before the decimal. This program is written using PLM for DOS. The current version is on \opstat\rbin.

UPDATE.PIF (Update ILC)

This utility is run after any changes are made to the database. It generates a new database and downloads it into the ILC selected by the user. To run UPDATE, choose UPDATE ILC from the Windows desktop and enter the ILC number. The batch file runs ILC.EXE, compiles and links the resulting plm code to produce an executable file called DBMAIN. DBMAIN is run to produce the ILC run file: dbxxxx.run. LOADILC is invoked to download the runfile. **NOTE**: to run this utility, you must have logged into the system. Otherwise, you do not have sufficient permissions to do the update.

Q&E Database Editor

Is a commercial database editor from Pioneer Software. It is used for direct editing of the offline database and generating queries. Since it provides no data validation, using it to change the database should only be done by experienced users.

DBGEN.EXE

A data entry program for the offline database written using the Clipper database compiler. (See Using DBGEN above).

dBASE IV

Is occasionally used for sorting the DEVICE and ILC database or for modifying database structures.

Glossary of Terms

active

The active area is the memory area in the ILC that is automatically and continuously updated to a matching area in the CMM. It contains data that can change dynamically like ADC readings.

ADC

Analog to Digital Converter. The ILC has 4 such channels. The old ILCs have 13 bit accuracy, the new ones have 16 bit accuracy.

BeamOrder

A unique number assigned to each device representing the devices relative position in the direction of the electron beam.This number increases from the Electron Gun to the Storage Ring. It is used as a key to relate a device to the channels used by the device.

BM

Boolean Monitor channel.

boolean

This name refers to a single wire control or monitor that can only have two states: on/off, open/closed etc. The ILC uses 0 or 255 for the two states (255 means opto high).

channel type

A channel is an input or output from the ILC. It can be an ADC, DAC, digital, GPIB, or serial type of access.

closed loop

The ILC contains an algorithm for making the Analog Monitor track the Analog Control. This feature can be turned on and off by changing the ClosedLoop field in the AC Active structure.

CMM

Collector Micro Module contains, in shared memory, the sum of all the ILC databases.

DAC

Digital to Analog Converter (AC). The ILC has 4 16 bit DAC chips for analog control.

DBELEMENT

The ILC database starts with an array of DBELEMENT structures. These contain the, names of the database entries which are the unique keys used to find a database entry.

device type

An entry in the device database and the Type field in the device channel that tells the number and type of channels used by the device.

devices

A device is a collection of logically related channels in an ILC database. A device usually corresponds to hardware like an interface chassis.

DMM

Display Micro Module. Has direct memory access to the CMM and serial access to the PC to handle display of accelerator data.

download

The action of loading files into an ILC. This is necessary after a change is made to the database. Downloading is done by the LOADILC utility.

EPICS

Experimental Physics and Industrial Control System. Originally developed at Los Alamos as an accelerator control system.

file server

All console and development PCs use one file server to contain shared files such as the database files. This PC is named light40.

group

File permissions on the file server are granted on the basis of the group that the user belongs to.

ID_NAME

Is the part of a database name that is used to identify the kind of device. These names have been standardized. For example, all quadrupole defocusing magnets begin with QD.

ILC

Intelligent Local Controller. A 80C186 based controller used to control accelerator hardware. Each ILC reports its data back to the CMM on a serial link.

ilc number

The unique address of an ILC. Values range from 1 to 999, with 1-63 reserved. Subranges of ILC numbers identify the location within a subsystem of the ILC.

inactive

The inactive area is the memory area in the ILC that contains data that does not change dynamically, like a name. This data is uploaded to the CMM only after a boot of the ILC.

offline database

The collection of dBASE files stored on the file server that contain the data needed to build the ILC database.

Operating State

The state that a boolean value normally has when the accelerator is operating normally.

PLM

A language for programming the 8086 family of processors. It is used in the ILCs, DMM, and on DOS.

pseudo-channel

This is a database channel, that is not associated with a real hardware channel. For example an AM structure could be used by ILC code just to store a calculated float value.

ready

The name given to a boolean monitor channel that is tells whether a device is ready to be activated. This should be the sum of all the conditions in the chain including the ILC remote/local monitor.

Run file

One of the binary files loaded into the ILC.

SBX module

This is a module that can be plugged into the industry standard SBX connector on the ILC to extend the ILC's control capability.

send buffer

There is a set of buffers in the CMM, one per ILC that is used to queue messages to be sent to the ILC (e.g. setpoints).

software seal

The name of an algorithm that is run in the ILC that insures that a boolean control is turned off when the ready monitor goes to the not-ready state.

Updating

Updating an ILC is the process of recreating the dbxxxx.run file from the Offline database and downloading into the ILC.

Index

U

V